Examen de Programación Concurrente - Clave a

Febrero 2009

Departamento de Lenguajes, Sistemas Informáticos e Ingeniería del *Software*

Normas

Este examen es un cuestionario tipo test que consta de **10 preguntas** en **5 páginas**. La puntuación total del examen es de **10 puntos**. La duración total es de **una hora y media**. El examen debe contestarse en las **hojas de respuestas**. No olvidéis rellenar **apellidos, nombre y DNI** en cada hoja de respuesta.

Sólo hay una respuesta válida por pregunta. Toda pregunta en que se marque más de una respuesta se considerará incorrectamente contestada. Toda pregunta incorrectamente contestada restará del examen una cantidad de puntos igual a la puntuación de la pregunta dividido por el número de alternativas ofrecidas en la misma.

La solución al examen se proporcionará antes de la revisión. Las calificaciones se darán a conocer el **17 de febrero**. La revisión del examen tendrá lugar el **19 de febrero**.

Cuestionario

(1 punto) 1. Dada la siguiente propiedad de un programa concurrente con dos procesos A y B: "*Alguna vez el proceso A ejecuta la sección crítica sin que el proceso B esté ejecutando código de la sección crítica.*"

Se pide marcar la afirmación correcta.

(a) Es una propiedad de seguridad.

(b) Es una propiedad de vivacidad.

(c) No es una propiedad.

(d) Ninguna de las otras respuestas es correcta.

(1 punto) 2. El siguiente tipo de tareas `T` implementa un protocolo de acceso a una sección crítica[1]:

```
task type T (I : PID);

task body T is
begin
   loop
      S;
      while Turno /= I loop null; end loop;
      Seccion_Critica;
      Turno := Turno + 1;
   end loop;
end T;
```

Dado un programa concurrente con `MAX_TASKS` tareas de tipo `T` compartiendo una variable `Turno` inicializada a `PID'First` y cada una de ellas con un índice `I` distinto.

Se pide marcar la afirmación correcta.

(a) El programa no cumple la propiedad de exclusión mutua en `Seccion_Critica`.

(b) Garantizada la terminación de `S` y `Seccion_Critica`, el programa no cumple la propiedad de ausencia de interbloqueo.

(c) Garantizada la terminación de `S` y `Seccion_Critica`, el programa no cumple la propiedad de ausencia de inanición.

(d) Garantizada la terminación de `S` y `Seccion_Critica` el programa cumple las propiedades de exclusión mutua en `Seccion_Critica`, ausencia de interbloqueo y ausencia de inanición pero un proceso podría esperar para ejecutar `Seccion_Critica` sin que los demás ejecuten `Seccion_Critica` ni compitan para hacerlo.

[1] Durante el examen se aclaró que la sentencia de incremento de `Turno` debía asumirse como `if Turno = MAX_TASKS then Turno := 1; else Turno := Turno + 1; end if` y que la atomicidad de la misma no es relevante para la respuesta

(1 punto) 3. Dado el siguiente programa concurrente (los semáforos deben considerarse inicializados tal y como indican los comentarios en su declaración):

```
X : Integer := 0;
S1 : Bin_Semaphore; -- := 1
S2 : Bin_Semaphore; -- := 0

task A;
task B;
task C;

task body A is
begin
   Wait (S2);
   Wait (S1);
   X := 2 * X;
   Signal (S1);
end A;
```

```
task body B is
begin
   Wait (S1);
   X := X * X;
   Signal (S1);
end B;

task body C is
begin
   Wait (S1);
   X := X + 3;
   Signal (S2);
   Signal (S1);
end C;
```

Se pide marcar el conjunto que contiene únicamente los posibles valores de la variable `X` tras la terminación de las tareas `A`, `B` y `C`.

(a) $\{3, 6, 9, 18, 36\}$

(b) $\{3, 6, 18, 36\}$

(c) $\{6, 18, 36\}$

(d) No está garantizada la terminación de todas las tareas.

(1 punto) 4. La condición de sincronización de una operación de un recurso compartido depende de un dato de entrada en un intervalo de datos entre `1` y `N`. Dicha operación va a ser invocada sólo por un proceso.

Se pide señalar la afirmación correcta.

(a) La implementación de la operación mencionada mediante objetos protegidos en Ada exige introducir una familia de entradas indexada con el tipo `1 .. N`

(b) La implementación de la operación mediante objetos protegidos en Ada es viable introduciendo una única entrada aplazada extra (al margen de cómo se implementen el resto de las operaciones).

(c) Si `N` fuera un número muy grande, la implementación de dicha operación mediante objetos protegidos en Ada exigiría introducir una familia de entradas indexada por el número de procesos (aunque este número sea 1).

(d) Ninguna de las otras respuestas es correcta puesto que no nos encontraríamos ante un programa concurrente.

(1 punto) 5. Según G. R. Andrews y F. B. Schneider, "[...] si un canal de comunicación tiene una capacidad de almacenamiento ilimitada, en la práctica permite que los procesos que ejecutan un *send* nunca queden bloqueados y a este tipo de paso de mensajes se le denomina asíncrono. [...] En el otro extremo, cuando el proceso que invoca un *send* queda bloqueado hasta que el correspondiente *receive* sea ejecutado, se habla de paso de mensajes síncrono."

Se pide marcar la afirmación correcta (ignorando limitaciones de almacenamiento de datos en tiempo de ejecución).

(a) No es posible implementar paso de mensajes síncrono utilizando paso de mensajes asíncrono.

(b) No es posible implementar paso de mensajes asíncrono utilizando paso de mensajes síncrono.

(c) Es posible implementar paso de mensajes síncrono utilizando paso de mensajes asíncrono sin introducir nuevos procesos en el sistema.

(d) Es posible implementar paso de mensajes asíncrono utilizando paso de mensajes síncrono sin introducir nuevos procesos en el sistema.

(1 punto) 6. Dadas las definiciones de paso de mensajes síncrono y paso de mensajes asíncrono de la pregunta 5 y dada la siguiente implementación de canales:

```
protected type Channel is
   entry Send (M : in Message);
   entry Receive (M : out Message);
private
   Empty : Boolean := True;
   Data : Message;
   Sender_Waiting : Boolean := False;
   entry Send_Delayed;
end Channel;

protected body Channel is
   entry Receive (M : out Message)
   when not Empty is
   begin
      M := Data;
      Empty := True;
   end Receive;
   entry Send (M : in Message)
   when Empty and not Sender_Waiting is
   begin
      Data := M;
      Empty := False;
      Sender_Waiting := True;
      requeue Send_Delayed;
   end Send;

   entry Send_Delayed when Empty is
   begin
      Sender_Waiting := False;
   end Send_Delayed;
end Channel;
```

Se pide marcar la afirmación correcta.

(a) Implementa paso de mensajes síncrono.

(b) No implementa ni paso de mensajes síncrono ni paso de mensajes asíncrono.

(c) Implementa paso de mensajes síncrono y paso de mensajes asíncrono.

(d) Implementa paso de mensajes asíncrono.

(1 punto) 7. Se pretende que una tarea que ejecute el código `loop ...; Esperar(R); ...; end loop;` se detenga en la llamada a la operación `Esperar` hasta que se cumpla una condición `Cond`. La llamada se ha descompuesto en el cliente (procedimiento `Esperar`) e implementado en el servidor (tarea `R`) utilizando *rendez-vous* y paso de mensajes síncrono de la siguiente forma:

```
procedure Esperar (R : Recurso);
   C : Channel;
begin
   Create (C);
   R.Esperar (C);
   Send (C, False);
   Destroy (C);
end Esperar;
```

```
-- Código del servidor R
select
   when Cond =>
      accept Esperar (C: in out Channel) do
         Aux := C;
      end Esperar;
      Receive (Aux, V);
or
   ...
end select;
```

Se pide marcar cuál de las siguientes afirmaciones es la correcta.

(a) Los procesos se bloquean porque el cliente debe hacer un `Receive` y el servidor un `Send`. Es decir, deben intercambiarse las llamadas que hacen ahora.

(b) Funciona, sólo se bloquea el cliente cuando la condición `Cond` no se cumple.

(c) Se bloquea, pero funcionaría si el cliente envía un `True` en lugar de un `False` para hacer cierta la condición `Cond`.

(d) El código produce siempre un interbloqueo del cliente y el servidor.

(1 punto) 8. **Nota:** pregunta anulada en el examen debido a la existencia de dos respuestas correctas (es muy interesante observar que las respuestas correctas e incorrectas pueden ser deducidas independientemente de la especificación del recurso).

A continuación se muestra un diseño de un sistema concurrente con una especificación formal de un recurso compartido (no se muestra el interfaz pero no contiene otras operaciones que las especificadas siendo el segundo argumento de las operaciones de tipo $\mathbb{Z}$) y tres tareas `T1`, `T2` y `T3` que comparten dicho recurso (variable `N : Notificacion`).

TIPO: $Notificacion = \mathbb{Z}$
INICIAL(n)**:** $n = 0$

CPRE: Cierto

Notificar(n,x)
POST: $n^{sal} = x$

CPRE: $n \neq x$

Sincronizar(n,x)
POST: $n^{sal} = n^{ent}$

```
task body T1 is
begin
   Notificar (N, 1);
end T1;

task body T2 is
begin
   Sincronizar (N, 0);
   Put (Integer'Image (0));
end T2;

task body T3 is
begin
   Sincronizar (N, 1);
   Put (Integer'Image (1));
end T3;
```

Se pide marcar la afirmación correcta una vez que el programa ha terminado y suponiendo que las operaciones `Put` son atómicas.

(a) "`0`" no es una salida posible del programa.
(b) "`1`" no es una salida posible del programa.
(c) "`01`" no es una salida posible del programa.
(d) "`10`" no es una salida posible del programa.

(1 punto) 9. Dada la siguiente implementación del recurso del problema 8:

```
protected type Notificacion_OP is
   entry Notificar (X : in Integer);
   entry Sincronizar (X : in Integer);
private
   N : Integer := 0;
   Sincronizado : Boolean := True;
   Copia_De_X : Integer;
   entry Sincronizar_Aplz
     (X : in Integer);
end Notificacion_OP;

protected body Notificacion_OP is
   entry Notificar (X : in Integer)
   when True is
   begin
      N := X;
   end Notificar;

   entry Sincronizar (X : in Integer)
   when Sincronizado is
   begin
      Copia_De_X := X;
      Sincronizado := False;
      requeue Sincronizar_Aplz;
   end Sincronizar;

   entry Sincronizar_Aplz
     (X : in Integer)
   when N /= Copia_De_X is
   begin
      Sincronizado := True;
   end Sincronizar_Aplz;
end Notificacion_OP;
```

Se pide marcar la afirmación correcta.

(a) Es una implementación correcta del recurso compartido.
(b) Pueden desbloquearse procesos que no deben (violación de una condición de sincronización).
(c) Ninguna de las otras respuestas es correcta.
(d) Hay que crear una familia de entradas indexada por `Integer`.

(1 punto) 10. Dada la siguiente implementación del recurso del problema 8[2]:

```
type Info_Sinc is record
   X : Integer;
   C : CB.Channel;
end record;

package Colas_Sinc is
   new Colas (Info_Sinc);
use Colas_Sinc;

procedure Notificar
  (N : in out Notificador;
   X : in     Integer) is
begin
   N.Notificar (X);
end Notificar;

procedure Sincronizar
  (N : in out Notificador;
   X : in     Integer) is
   C : CB.Channel;
   B : Boolean;
begin
   CB.Create (C);
   N.Sincronizar (X, C);
   CB.Receive (C, B);
   CB.Destroy (C);
end Sincronizar;

task body Notificador_RV is
  N : Integer := 0;
  N_Por_Sinc : Natural := 0;
  Por_Sinc : Cola;
begin
  Crear_Vacia (Por_Sinc);
  loop
    select
      when True =>
        accept Sincronizar
          (X : in Integer;
           C : in out CB.Channel) do
          Insertar (Por_Sinc, (X, C));
          N_Por_Sinc := N_Por_Sinc + 1;
        end Sincronizar;
    or
      when True =>
        accept Notificar
          (X : in Integer) do
          N := X;
        end Notificar;
        declare
          Por_Atender : Natural := N_Por_Sinc;
          Pet : Info_Sinc;
        begin
          while Por_Atender > 0 loop
            Primero (Por_Sinc, Pet);
            Borrar (Por_Sinc);
            Por_Atender := Por_Atender - 1;
            if N /= Pet.X then
              CB.Send (Pet.C, True);
              N_Por_Sinc := N_Por_Sinc - 1;
            else
              Insertar (Por_Sinc, Pet);
            end if;
          end loop;
        end;
    end select;
  end loop;
end Notificador_RV;
```

Se pide marcar la afirmación correcta.

(a) Es una implementación correcta del recurso compartido.

(b) Pueden desbloquearse procesos que no deben (violación de una condición de sincronización).

(c) Ninguna de las otras respuestas es correcta.

(d) Pueden bloquearse procesos que no deben.

[2] **declare** D **begin** S **end;** es una sentencia de Ada que permite declarar variables (D) para un bloque de sentencias (S) y que provoca la ejecución de dichas sentencias.